Manual de Usuario
Drarch - Discover and Recover Architecture

{FRONTINI, FMALDONA}@EXA.UNICEN.EDU.AR

12 de octubre de 2006

Índice

1. Instalación **1**
1.1. Requerimientos 1
1.2. Plugins del Drarch 2
1.3. Instalación de Eclipse 2
1.4. Instalación de Plugins 2

2. Proceso **3**

3. Vista de Sugerencia **4**

4. Editor de Reglas **5**
4.1. Creación de una ragla 5

5. Generación de log´s **7**

6. Apéndice **7**
6.1. Predicados JQuery 7
6.1.1. Predicados unarios 9
6.1.2. Predicados binarios 10
6.1.3. Predicados ternarios 10
6.1.4. Predicados derivados 11
6.2. Predicados Drarch 13

1. Instalación

1.1. Requerimientos

Como la aplicación se desarrolló como un Plugin para Eclipse, es necesario tener el Eclipse 3.1.2 con los siguientes plugins instalados:

- EMF 2.1
- GEF 3.1
- Flabot (provisto por el Drarch ya que se realizaron modificaciones: se exportaron paquetes)
- JQuery (provisto por el Drarch ya que se realizaron modificaciones: se optimizó para aumentar la performance)

1.2. Plugins del Drarch

Los siguientes plugins corresponden a la herramienta Drarch. Estos deben ser copiados en la carpeta ECLIPSE/PLUGINS.

- Drarch 3.1
  - Drarch Plug-in 3.1
  - Drarch Diagram Plug-in 3.1
  - Drarch Engine Plug-in 3.1
  - Drarch JqueryImpl Plug-in 3.1
  - Drarch Ui Plug-in 3.1
  - Drarch Util Plug-in 3.1

1.3. Instalación de Eclipse

Sitios de descarga

- JDK de Sun: http://java.sun.com/javase/downloads/index.jsp
- Eclipse: http://download.eclipse.org/eclipse/downloads/

La instalación de eclipse debería hacerse preferentemente en una ruta de directorios que no contenga espacios, simplemente descomprimiendo el archivo .zip (o .tar.gz).

Si al tratar de ejecutar el Eclipse se queja por no encontrar la máquina virtual de java, es necesario agregar, ya sea en un shortcut o desde linea de comandos los argumentos '-vm path-a-java' siendo path-a-java la ruta al ejecutable javaw.exe (o su equivalente en linux).

1.4. Instalación de Plugins

Tanto el EMF como el GEF, se pueden instalar desde el propio Eclipse utilizando el Update Manager. Para activarlo vamos al menú Help ->Software Updates ->Find and Install. Ahí seleccionamos Search for new features to install.

EMF

En la vista de los sitios de instalación seleccionamos New Remote Site, ponemos EMF Update Site y http://download.eclipse.org/tools/emf/updates respectivamente y seleccionamos Finish. Seleccionamos el mirror por defecto (o sea presionamos OK). Destildamos la opción Show the latests version of a feature only y seleccionamos del árbol EMF SDK 2.1.2. Le damos Next, aceptamos la licencia, Next otra vez y luego Finish. Luego confirmamos que queremos instalar todo (Install All) y aceptamos que reinicie Eclipse.

GEF

El proceso es similar al EMF. Lo que varía es el update site y el componente a seleccionar.

Update Site:
nombre: GEF Update Site
url: http://update.eclipse.org/updates/3.0
El componente a seleccionar es GEF 3.1.1

Drarch

Se deben descomprimir los archivos Drarch.rar, Flabot.rar y JQuery.rar. Luego, copiar el contenido de cada archivo en el directorio donde se encuentra el eclipse. Los plugins se pueden bajar del sitio: http://sourceforge.net/projects/drarch

2. Proceso

Para iniciar el proceso de reconstrucción de la arquitectura, se debe tener un uno o más proyectos dentro del workspace, de esta manera, el desarrollador va a poder seleccionar el código a analizar.

Luego, por medio de la vista de sugerencias, se selecciona el código fuente a analizar (archivos .java) y se comienza con la primer etapa, la cual, muestra una lista de sugerencias. Estas sugerencias pueden ser aceptadas o rechazadas. Una vez que se terminó con la selección de sugerencias, se debe pasar a la próxima etapa. El paso de una etapa a la otra hace que se genere un nuevo diagrama de componentes (si es que en la etapa anterior se seleccionaron sugerencias). Este proceso se repite hasta que no se encuentren más reglas.

Luego de la última regla, se debe realizar el análisis del código en ejecución. Para esto, se debe utilizar el FLABOT para generar el log. El log generado debe representar un caso de uso del sistema, y debe ser lo suficientemente concreto para que se refleje en diagramas de UCMs. De esta forma, se deben hacer tantos log´s como casos de uso existentes.

A medida que se van generando los log´s, se pueden ir creando diagramas de UCMs. Para realizar esto, se debe utilizar el Drarch e indicarle cual es el archivo que se quiere analizar para que genere el UCM correspondiente.

En resume, el proceso debería seguir los siguientes pasos:

1. Seleccionar archivos .java

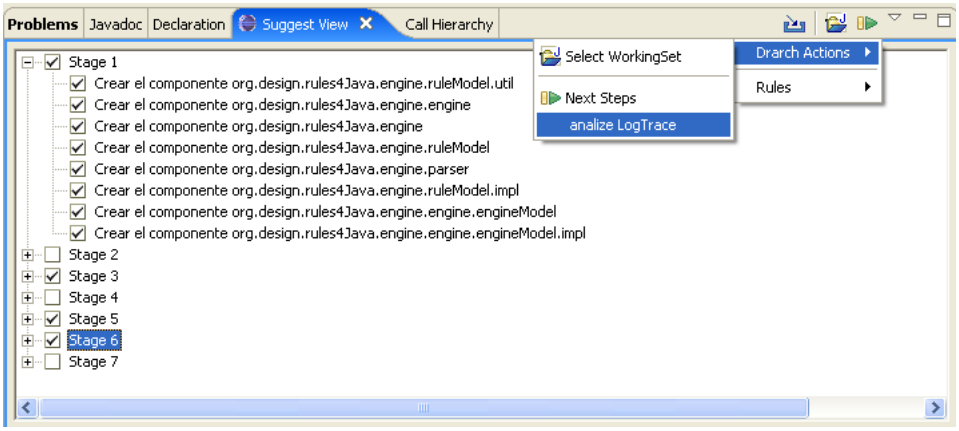

Figura 1: Vista de Sugerencias

2. Seleccionar el archivo de reglas (este punto se puede omitir si se desea utilizar las reglas por defecto)
3. Comenzar con la siguiente etapa
4. Seleccionar las sugerencias
5. Ir al paso 3 a menos de que no se encuentren más etapas.
6. Utilizar el FLABOT para generar archivos logs
7. Generar UCMs con los archivos logs generados en el paso anterior.

3. Vista de Sugerencia

Para abrir la vista de sugerencias se debe seleccionar la vista Suggest Viewe de la categoría Drarch.

En la figura1 se muestra la vista de sugerencias.

- Import Rule XML: permite importar un archivo XML con las reglas que se van a utilizar en el proceso de reconstrucción de la arquitectura. En caso de no importar ninguna regla, la herramienta utiliza reglas por defecto.
- Select workingset: permite seleccionar el proyecto a ser analizado (se deben seleccionar solo los .java)
- Next Step: Ejecuta cada regla (una por etapa) y una vez seleccionada la sugerencia, crea diagramas en el archivos flabot ubicado en la carpeta Drarch del proyecto seleccionado.

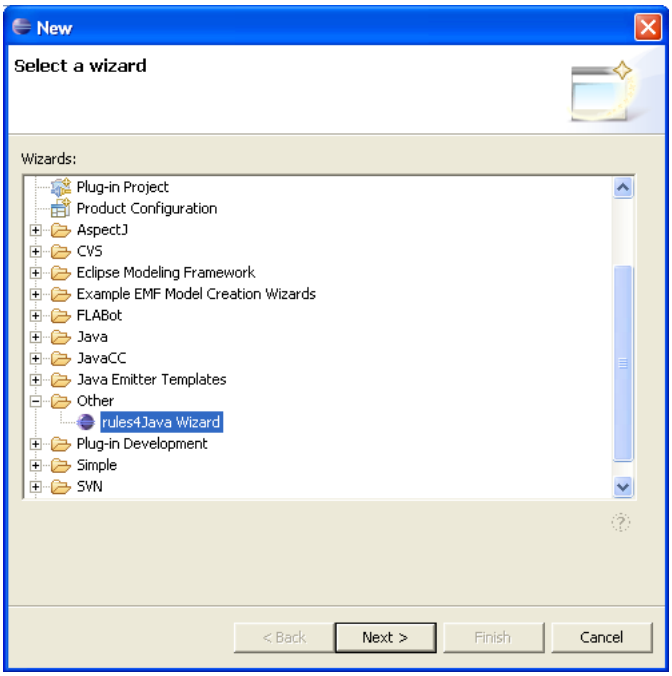

Figura 2: Selección de wizard Rules4Java

- analize LogTrace: Esta opción permite realizar un análisis del archivo de log generado durante la ejecución del sistema a analizar. Esta acción genera diagramas de UCMs.

4. Editor de Reglas

El drarch toma por defecto un conjunto de reglas, sin embargo, es posible crear un archivo de reglas y reemplazarlas para que sean usadas en vez de las reglas por defecto. Para crear un archivo de reglas se debe seleccionar el wizard de la figura 2 por medio de File ->New ->Other...

Luego, para poder agregar, eliminar y editar las reglas, se debe hacer doble click en el archivo creado por el wizard y de esta forma, se abre el editor de la figura 3.

4.1. Creación de una ragla

Una vez que se tiene el editor de reglas, los siguientes pasos describen la creación de una regla en particular.

1. Descripción breve de lo que hace la regla.
2. Agregar una consulta al estilo prolog.

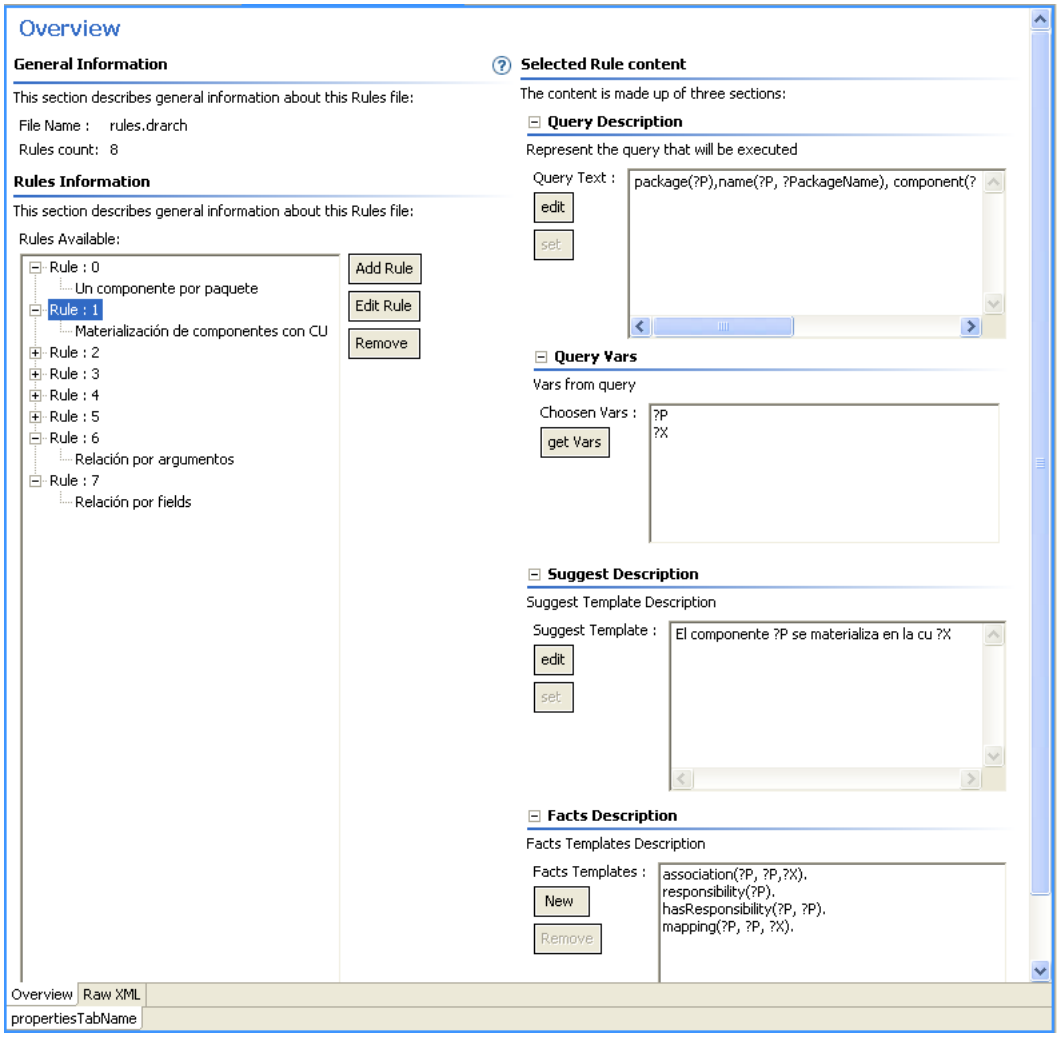

Figura 3: Editor de Reglas

3. Seleccionar las variables de la consulta que se desean obtener.
4. Generar un *template* que describa una sugerencia utilizando las variables seleccionadas en el punto anterior.
5. Crear nuevos hechos que se tendrían que incorporar a la base de conocimiento.

Ejemplo

Para poder ver este proceso un poco mas claro, se muestra el siguiente ejemplo en el cuál se intenta crear una regla que sugiera la creación de un componente por cada paquete del proyecto seleccionado.

1. La descripción de la regla podría ser: "Un componente por paquete".
2. La consulta sería: PACKAGE(?P)
3. La variable que se debe seleccionar es: ?P
4. El *template* es: Crear el componenten ?P.
5. El hecho que se agrega es: COMPONENT(?P).

5. Generación de log´s

Para poder generar los archivos log, se debe utilizar el plugin FLABOT. Dentro del wizard para las configuraciones del Run, se debe crear un nuevo FLABOT COLLECTING SESSION como se muestra en la figura 4.

Básicamente, se debe especificar cuál es el LAUNCH a analizar, el archivo en cual se va a crear el log y el archivo .flabot con los diagramas de componentes generados por el Drarch.

Luego, se ejecuta el FLABOT COLLECTING SESSION que se configuró. Ésto hace que se ejecute la aplicación que se está analizando para extraer la arquitectura. Se debe realizar un caso de uso específico sobre la aplicación. Una vez finalizado el caso de uso, se cierra la aplicación y se termina de loggear. Este proceso se repite variando los casos de uso.

6. Apéndice

6.1. Predicados JQuery

Los predicados de JQuery respetan la siguiente jerarquía:

- Element
  - Package
  - CU

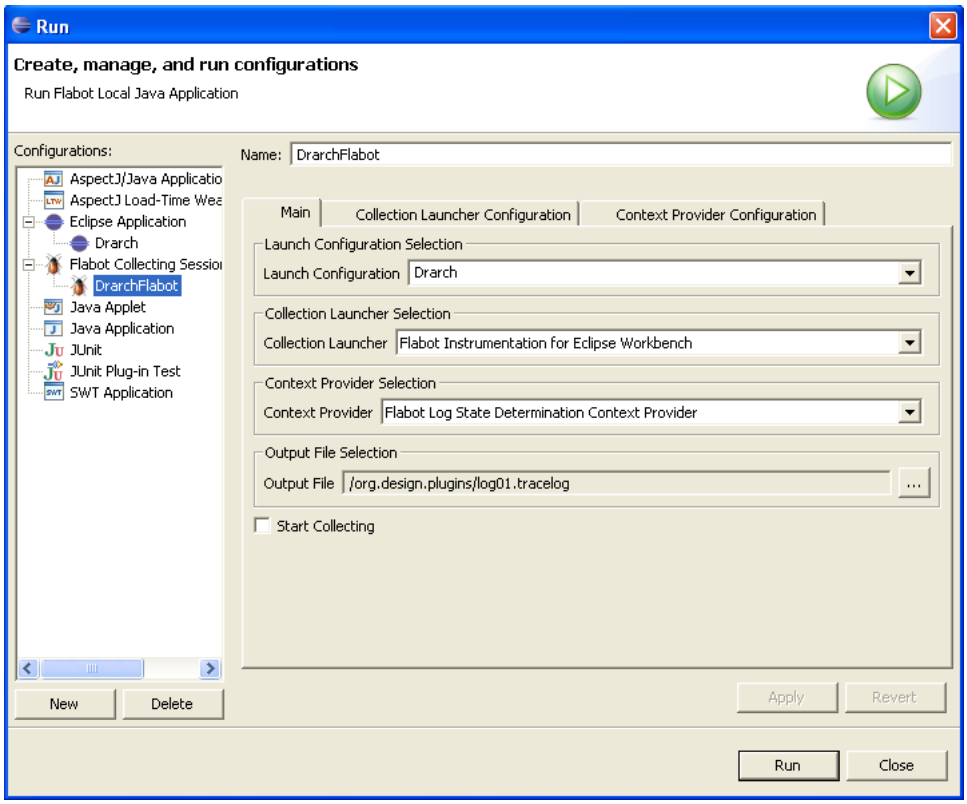

Figura 4: Flabot Collecting Session

- Field
- Type
  - Primitive
  - RefType (interface or class)
- Block
  - Initializer
  - Callable
    - Method
    - Constructor
- Marker
  - Bookmark
  - Warning
  - Error
  - Task

6.1.1. Predicados unarios

Predicate Name	Argument Type	Description
cu	CU	cu(?X) means: "?X is a Compilation Unit (.class or .java file)"
package	Package	package(?X) means: "?X is a package"
class	RefType	class(?X) means: "?X is a class"
interface	RefType	interface(?X) means: "?X is an interface"
method	Method	method(?X) means: "?X is a method"
constructor	Constructor	constructor(?X) means: "?X is a constructor"
initializer	Initializer	initializer(?X) means: "?X is an initializer"
field	Field	field(?X) means: "?X is a field"
bookmark	Bookmark	bookmark(?X) means: "?X is a bookmark"
warning	Warning	warning(?X) means: "?X is a compiler warning"
error	Error	error(?X) means: "?X is a compiler error"
task	Task	task(?X) means: "?X is a task"

6.1.2. Predicados binarios

Predicate Name	Argument Types	Description
priority	Task, String	priority(?T,?P) means: "Task ?T has priority ?P"
name	Element, String	name(?E,?S) means: "Element ?E has name ?S"
child	Element, Element	child(?Sup,?Sub) means: "Element ?Sup has a child ?Sub"
extends	RefType, RefType	extends(?C1,?C2) means: "Class (or Interface) ?C1 extends Class (or Interface) ?C2"
implements	RefType, RefType	implements(?C,?I) means: "Class ?C implements Interface ?I"
throws	Callable, RefType	throws(?C,?T) means: "Callable ?C throws ?T"
type	Field, Type	type(?F,?T) means: "Field ?F is of type ?T"
modifier	Element, String	modifier(?E,?S) means: "Element ?E has modifier (i.e public, private, static, etc) ?S"
arg	Callable, Type	arg(?C,?T) means: "Callable ?C has an argument of type ?T"
returns	Callable, Type	returns(?C,?T) means: "Callable ?C returns Type ?T"
signature	Callable, String	signature(?C,?S) means: "Callable ?C has signature ?S"

6.1.3. Predicados ternarios

Predicate Name	Argument Types	Description
methodCall	Block, Method, SourceLocation	methodCall(?B,?M,?L) means: "Block ?B calls Method ?M at location ?L"
thisCall	Constructor, Constructor, SourceLocation	methodCall(?C1,?C2,?L) means: "Constructor ?C1 makes a "this" call to Constructor ?C2 at location ?L"

superCall	Callable, Callable, SourceLocation	superCall(?C1,?C2,?L) means: "Callable ?C1 makes a "super" call to Callable ?C2 at location ?L"
constructorCall	Block, Constructor, SourceLocation	constructorCall(?B,?C,?L) means: "Block ?B calls Constructor ?C at location ?L"
instanceOf	Block, RefType, SourceLocation	instanceOf(?B,?T,?L) means: "Block ?B performs an instanceof test for type ?T at location ?L"
reads	Block, Field, SourceLocation	reads(?B,?F,?L) means: "Block ?B reads field ?F at location ?L"
writes	Block, Field, SourceLocation	writes(?B,?F,?L) means: "Block ?B writes to field ?F at location ?L"
param	Callable, Type, Integer	param(?C,?T,?N) means: "Callable ?C has an argument of type ?T as its ?Nth argument"
tag	Element, String, String	tag(?E,?N,?V) means: "Element ?E has javadoc tag ?N with value ?V"

6.1.4. Predicados derivados

Predicate Name	**Argument Types**	**Description**
type	Type	type(?T) means: "?T is a type (a class, interface, or primitive)"
element	Element	element(?E) means: "?T is an element"
likeThis	Element, Element	likeThis(?E1,?E2) means: "?E1 and ?E2 have the same name, but are not the same object"
strongLikeThis	Callable, Callable	strongLikeThis(?C1,?C1) means: "?C1 and ?C2 have the same signature"
child+	Element, Element	child+(?E1,?E1) means: "?E2 has ?E1 as one of its ancestors"

package	Element, Package	package(?E,?P) means: "Element ?E is in package ?P"
constructor	Class, Constructor	constructor(?CL,?CON) means: "Class ?CL declares constructor ?CON"
method	Type, Method	method(?T,?M) means: "Type ?T declares method ?M"
subtype	Type, Type	subtype(?T1,?T2) means: "?T2 is a direct subtype of ?T1"
subtype+	Type, Type	subtype+(?T1,?T2) means: "?T2 is in ?T1's type hierarchy"
subtype*	Type, Type	subtype*(?T1,?T2) means: "?T2 is in ?T1's type hierarchy (?T2 can equal ?T1)"
field	Type, Field	field(?T,?F) means: "Type ?T declares field ?F"
inheritedField	Type, Field, Type	inheritedField(?T,?F,?InhT) means: "Type ?T inherites field ?F from type ?InhT"
calls	Block, Callable, SourceLocation	calls(?B,?C,?L) means: "Block ?B calls ?C at location ?L"
staticCall	Block, Callable, SourceLocation	staticCall(?B,?C,?L) means: "Block?B statically calls ?C at location ?L"
dynamicCall	Block, Callable, SourceLocation	dynamicCall(?B,?C,?L) means: "Block ?B dynamically calls ?C at location ?L"
polyCalls	Block, Callable, SourceLocation	polyCalls(?B,?C,?L) means: "Block ?B calls polymorphic callable ?C at location ?L"
accesses	Block, Field, SourceLocation	accesses(?B,?F,?L) means: "Block ?B accesses field ?F at location ?L"
inheritedMethod	Type, Callable, Type	inheritedMethod(?T,?C,?SUP) means: "Type ?T inherits ?C from ?SUP"
overrides	Callable, Callable	overrides(?C1,?C2) means: "Callable ?C1 overrides callable ?C2"
creator	Class, Block, SourceLocation	creator(?C1, ?Ctor, ?L) means: "Class ?C1 is created by block ?Ctor at location ?L"

implements+	Class, Interface	implements+(?C,?I) means: "Interface ?I is implemented by ?C or an ancestor of ?C"
outerType	Type	outerType(?T) means: "?T is an outer type"
nestedType	RefType	nestedType(?T) means: "?T is a nested type"
memberType	RefType	memberType(?T) means: "?T is a member type"
staticNestedType	RefType	nestedType(?T) means: "?T is a static nested type"
localType	RefType	localType(?T) means: "?T is a local type"
anonymousType	RefType	anonymousType(?T) means: "?T is an anonymous type"
viewFromHere	Element, [Element]	viewFromHere(?X,-?ViewStructure) means: "?ViewStructure is a tree representation of all child relationships below parent element ?X"
re_match	RegExp, String	re_match(?RE,?S) means: "String ?S matches Regular Expression ?RE"
re_name	Element, RegExp	re_name(?E,?RE) means: "?E has a name that matches ?RE"

6.2. Predicados Drarch

Los predicados que se agregaron a la base de conocimiento (KnoledgeBase), representan conceptos de diagramas de componentes para representar arquitecturas.

Predicate Name	**Argument Type**	**Description**
component	String	component(?X) means: "?X is a Component"
interfaceLink	String, String	intefaceLink(?X, ?Y) means: "InterfaceModel ?X is linked with InterfaceModel ?Y"
interfaceModel	String	interfaceModel(?X) means: "?X is a Interface"
port	String	port(?X) means: "?X is a Port"
hasPort	String, String	hasPort(?X, ?Y) means: "Component ?X has Port ?Y"

hasProvidedInterface	String, String	hasProvidedInterface(?X, ?Y) means: "Port ?X has a provided interfaceModel ?Y"
hasRequiredInterface	String, String	hasRequiredInterface(?X, ?Y) means: "Port ?X has a required interfaceModel ?Y"
association	String, String, String	association(?X, ?Y, ?Z) means: "JavaFile ?Z is associated by the Component ?X in the package ?Y "
mapping	String, String, String	mapping(?X, ?Y, ?Z) means: "JavaFile ?Z is mapped by the Component ?X in the package ?Y "
relationship	String, String, String	relationship(?X, ?Y, ?Z) means: "Component?X has a relationchip with Component ?Y with the Prototype ?Z "
responsibility	String	responsibility (?X) means: "?X is a Responsability"
hasResponsibility	String, String	hasResponsibility (?X, ?Y) means: "Component ?X has a Responsibility ?Y "